Network performance of the JBoss Application Server

Benothman, Nabil ; Clere, Jean-Frederic ; Schiller, Eryk ; Kropf, Peter ; Maucherat, Remy

Abstract: JBoss Application Server (AS) uses java.io and the Apache Portable Runtime (APR) project to provide its HTTP connectors. Due to new features of upcoming specifications of the Java Enterprise Edition (Java EE), the existing connectors shall be replaced by modern non blocking Input/Outputs (I/Os). In this study, we review some modern I/O frameworks such as NIO.2 introduced by Java SE 7 and XNIO3 developed by JBoss. We compare their network performance by running a series of stress tests on client-server applications of limited functionality. As a result, we select NIO.2 as the most appropriate framework to specify and implement a new JBoss connector. Finally, we compare our newly implemented Java connector against the existing APR-based one by means of network performance measures.

DOI: https://doi.org/10.1109/LCN.2013.6761324

Network Performance of the JBoss Application Server

Nabil Benothman *†, Jean-Frederic Clere†, Eryk Schiller*, Peter Kropf*, and Rémy Maucherat†
* University of Neuchâtel, Neuchâtel, Switzerland
{nabil.benothman, eryk.schiller, peter.kropf}@unine.com
† RedHat, Inc., Neuchâtel, Switzerland
{jclere, rmaucher}@redhat.com

***Abstract*—JBoss Application Server (AS) uses java.io and the Apache Portable Runtime (APR) project to provide its HTTP connectors. Due to new features of upcoming specifications of the Java Enterprise Edition (Java EE), the existing connectors shall be replaced by modern non blocking Input/Outputs (I/Os). In this study, we review some modern I/O frameworks such as NIO.2 introduced by Java SE 7 and XNIO3 developed by JBoss. We compare their network performance by running a series of stress tests on client-server applications of limited functionality. As a result, we select NIO.2 as the most appropriate framework to specify and implement a new JBoss connector. Finally, we compare our newly implemented Java connector against the existing APR-based one by means of network performance measures.**

I. Introduction

In this paper, we focus on the JBossWeb component that implements web connectors for the JBoss AS [1], which is almost entirely written in Java. The main role of a web connector is to provide JBossWeb with an HTTP-based communication layer as presented in Figure 1. JBossWeb uses Tomcat's APR based web connector per default (c.f., Figure 2), which provides the best performance in comparison to other pure Java based connectors implemented to date. APR includes access to advanced I/O features in blocking and non blocking mode, however, it also brings portability issues as APR is not Java based, so it has to be separately compiled for all the supported hardware architectures. Since Java SE 7, there exists a pure blocking and non-blocking I/O framework called NIO.2. Also JBoss implements another similar I/O called XNIO3. Because both NIO.2 and XNIO3 provide us with synchronous and asynchronous communications of high performance, there is a new area for research as we can try to design and implement a pure Java based connector (c.f., Figure 2), which is able to perform as well as APR, whilst not needing to maintain several binaries for different architectures.

This paper is organized in the following way. In Section II, we compare NIO.2 against XNIO3 in blocking and non-blocking modes. We then select the I/O framework to implement a new JBoss connector. Section III quickly addresses some implementation details of our NIO.2 based JBoss connector and studies the performance of the JBoss AS with the new connector integrated. Section IV surveys previous work in the performance analysis of Java I/O frameworks and Web Servers. Finally, we conclude in Section V.

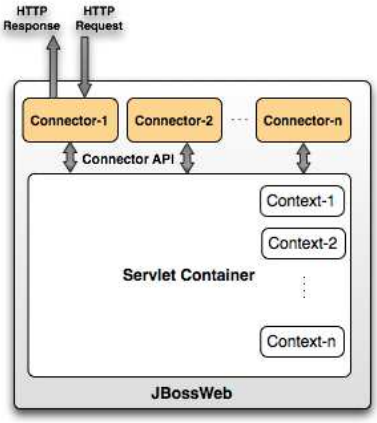


Fig. 1: JBoss Web Architecture.

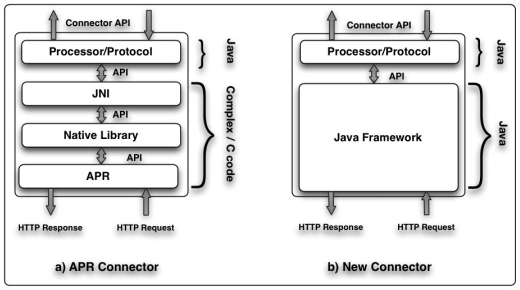


Fig. 2: JBoss Web Connector Architecture.

II. I/O Selection

At the beginning, we implement a small client and server in Java. The client uses an ordinary *java.io.Socket* I/O API, and it does not depend on NIO.2 or XNIO3. The client application starts N threads, and each thread maintains a single connection to the server, hence there are N simultaneous client connections. Each running thread periodically asks the server for content and waits until the response arrives. The requests we use resemble HTTP messages, but they only contain a single request line. The task of the server relies on delivering a static text file stored on the disk to the client. The server implementation uses NIO.2 or XNIO3 for connection handling. To reduce the server logic, the client requests are

not processed, i.e., the server immediately sends a static text file in response to the received query. Our server uses one thread per request instead of one thread per connection, because the latter does not scale well and does not fit the new servlet specification. The former allocates a thread for each incoming request, so when the request processing is finished, the thread is able to immediately handle another job. The latter is considered less appropriate, because it allocates one thread per connection, which remains blocked until the corresponding connection is closed. As an example, let us imagine a connected client, which sends one request per a few minutes. The thread has to be running throughout the whole life-cycle of the connection, however, it is almost idle, which results in waste of resources. Finally, the thread per connection model does not allow us to serve the same number of clients as the thread per request model due to resource limitations.

Our tests of NIO.2 and XNIO3 are based on important metrics reflecting the overall performance of the system. The main metric is the Response Time (RT), which is defined as the propagation delay between issuing a request and receiving the corresponding reply. For example, if a web server has very long RTs, the user may consider the service as broken and stop using it. Since the final goal of this project is to implement a web connector, the response time is highly significant, because it provides a better quality of service (QoS). The response time metric is measured on the client side, and it is collected by a measuring logic of the client application. We also measure the CPU consumption and memory usage on the server.

A. Test conditions

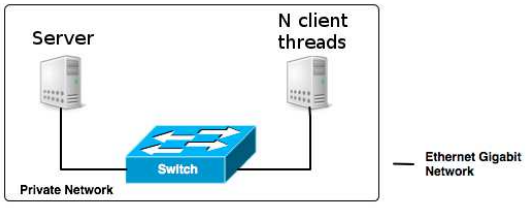


Fig. 3: Network setup for NIO.2 vs XNIO3.

We keep the same test conditions for both NIO.2 and XNIO3. Some of them are hardware and others are software or configuration related. To run our server and client applications, we use a laboratory environment with 2 64bit 8-core machines (the processor belongs to the x86_64 family and the clock rate is 3 GHz) with 4 GB RAM, where one machine is assigned to the server and the other to the client. These machines are physically connected to the switch through the Gigabit Ethernet network of link capacity around 100 MB/sec (c.f., Figure 3). Each machine runs the RHEL-4 operating system with Java Virtual Machine 7 to run our applications. The number of simultaneous client connection (threads) is 1000, while the response size of the server is 32'000 bytes.

B. Testing

In each test, we first set the fixed request load (the total number of requests per second). We then send a large number of requests between our client and server. For example, when the load is 1000 req/s, the client application sends 1000 requests in a second, which simply means that one thread is responsible for issuing 1 req/s. Subsequently, during the test we collect the memory consumption and the CPU load on the server, and the RT of every issued request. Figure 4 illustrates the probability of a given response delay for two tests (1000 req/s, 3500 req/s) in the case of synchronous NIO.2 after sending 100'000 requests. Axes for 1000 req/s reside on the top-right, while the plot for 3500 req/s uses bottom-left. To approximate the delay distribution, we fit Gamma functions $f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\beta x)$ by estimating α and β, while for low request rates (low delays), it was difficult to fit other functions, e.g., the Gaussian fit was unrealistic giving negative RTs in some cases. We can then quickly compute mean value

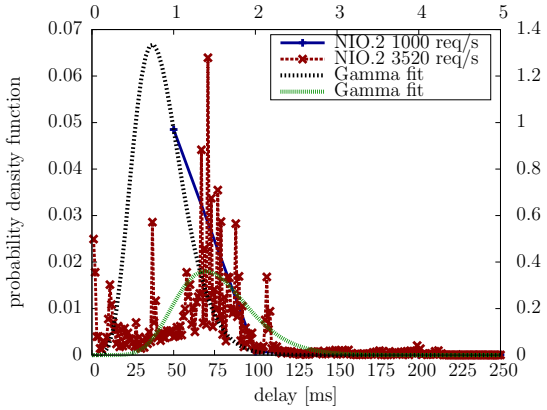


Fig. 4: Delay distribution in example nio2 synchronous tests.

$EX = \frac{\alpha}{\beta}$ and standard deviation $\sigma = \sqrt{\frac{\alpha}{\beta^2}}$ for every fitted Gamma function. In Figure 5, we plot these mean delays against different load values. We also equip every data point with an error bar of length equal to the corresponding standard deviation. By varying the request load, we can estimate the

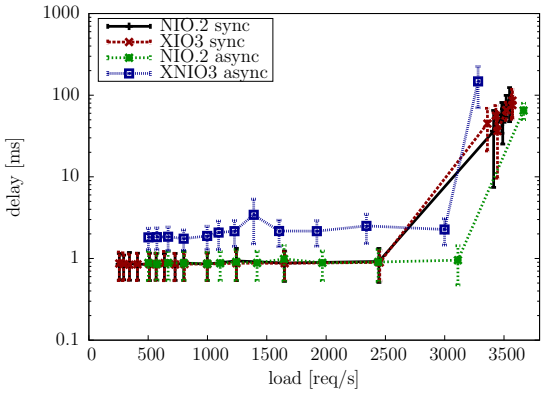
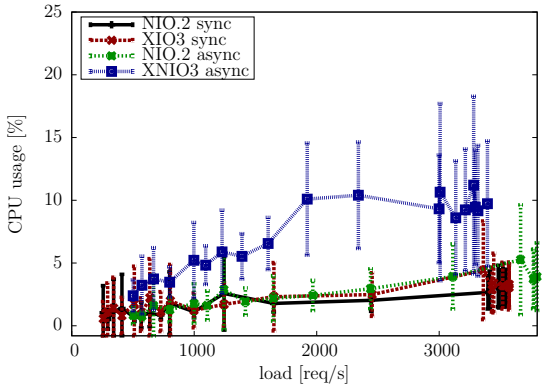
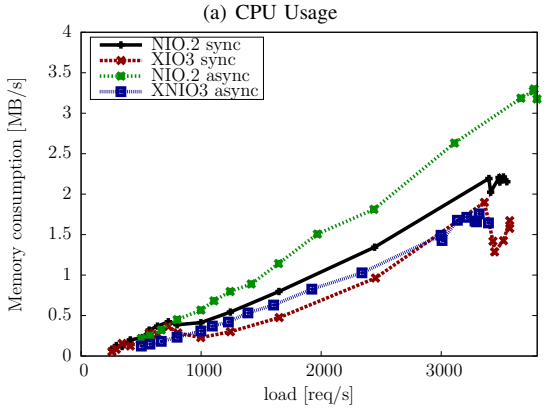


Fig. 5: Delay against load in nio2 vs xnio3 tests.

RT, CPU, and memory consumption as its function. At the



(a) CPU Usage



(b) Memory Consumption

Fig. 6: Other metrics measured on the server.

beginning, the response time does not change until the load arrives close to the network capacity of around 100 MB/s (3100 req/s) as we can see in Figure 5. In the blocking mode, NIO.2 and XNIO3 have similar RTs, while the non-blocking XNIO3 takes a little bit more time to handle requests in comparison to non-blocking NIO.2. Far away from the saturation point, the XNIO3 adds constant overhead of around 1 ms, however, this difference grows as the load approaches the network capacity.

In Figure 6(a), we can easily see that in the non-blocking mode XNIO3 loads a CPU 50% more than NIO.2. On the other hand, in the blocking mode, they both consume almost the same amount of CPU, because the blocking operations of NIO.2 and XNIO3 are equivalent to NIO.1. We also observe how the size of the server JVM process grows over time. In Figure 6(b), we can see that the non-blocking NIO.2 allocates almost 50% more memory than the other mechanism. This might be considered as a disadvantage from the performance point of view, because a high memory allocation to unreferenced objects may require the Java Garbage Collector (GC) to clean the memory when a targeted memory consumption level is reached. The GC has the highest priority, so all other java applications have to stop when the GC is running.

C. Technology Selection

As JBoss AS 7 can work under heavy workloads, we have decided to equip it with an NIO.2 based web connector for the following reasons: a) NIO.2 uses less CPU than XNIO3 in the non-blocking mode; and b) non-blocking XNIO3 generates longer RTs. The highly loaded environment requires special measures and the previously mentioned factors are crucial for the server performance.

III. Web Connector

In this section, we quickly describe implementation details of our newly developed web connector for the JBossWeb module. Our implementation is based on two layers as previously explained in Figure 2. It obviously includes all the features of the APR web connector. The JBossWeb module requires implementing two channel types: *org.apache.tomcat.util.net.NioChannel* and *org.apache.tomcat.util.net.jsse.SecureNioChannel*, which were originally developed by using the APR library. During this work, we have equipped them with an asynchronous read listener based on NIO.2, which waits in a non-blocking way until new data arrives. We mainly use it to handle non-blocking *wait* for reading client data without sacrificing corresponding threads. To communicate with the Servlet Container (which runs servlets), the web connector raises events such as READ, WRITE, TIMEOUT, END, and ERROR. E.g., a READ is issued when the JBossWeb asynchronously reads and new information appears in the input buffer designated to the web application.

A. Test setup

To compare APR against our newly developed web connector, we use a similar setup as in the case of the NIO.2 vs XNIO3 tests performed in Section II. We use the same metrics such as the RT and CPU usage, however, we do not consider the server's memory consumption, as JVM is always close to its limit (2 GB). We also fix the maximum number of threads on the server to 512 (the number of available processors multiplied by 64). Since we test real web applications, the client application sends valid HTTP requests and of course receives valid HTTP replies. The client application, however, does not process any received messages. The total number of clients connected to the server at the same time is 5000. Since the number of clients is too high to run on the same machine, we use 6 machines in total; one to run the application server and five others to run clients (1000 clients per machine). We also test high load, so to eliminate network concerns such as the network capacity, the size of the server response is less than 0.5 KB. Even with the highest load, the total amount of data transfers between the application server and clients is still much lower than the network capacity of around 100 MB/s.

B. Tests

Figure 8 presents the average server response delay as a function of load. According to this Figure, the NIO.2 web connector outperforms APR in both situations: blocking

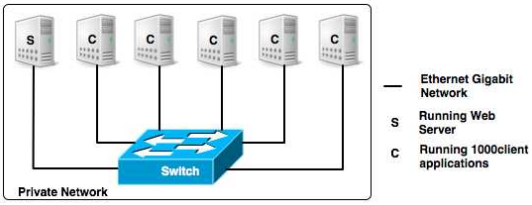


Fig. 7: Network setup for JBoss AS 7 tests with APR and NIO.2 Connectors.

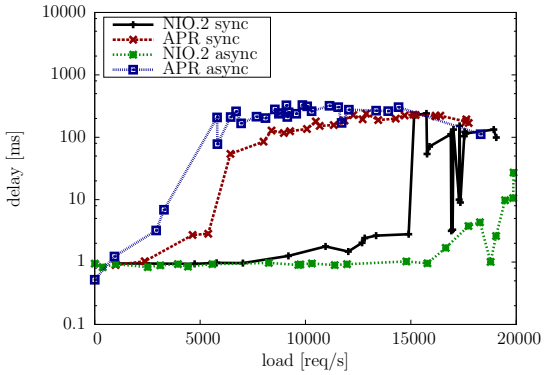


Fig. 8: Delay against load in JBoss AS 7, apr vs nio2 tests.

and non-blocking. This difference may be explained by a significant number of adaptation layers between the JVM and APR. In Figure 9, we observe the CPU consumption of the server against its load. In both situations our NIO.2 connector outperforms APR in terms of the CPU usage.

IV. Related Work

There exist many frameworks for evaluating different web services, however, the subject of evaluating different asynchronous Java I/Os in the case of the Java based AS has not been tackled to date. *Banga et al.* [2] provide a method of bursty traffic generation to measure the performance of a generic web server. *Hu et al.* [3] studied bottlenecks of different web servers. They identified many causes of low performance such as lack of caching, protocol processing of HTTP messages, I/Os, and concurrency strategies (thread-pool model, thread per request model); they used their knowledge to improve the JAWS web server. *Cane* [4] provides a laboratory setup for testing performance of web applications by measuring the latency, throughput, and memory consumption under different loads. His measuring techniques are similar to our methods. In the past, the performance of JAVA I/Os was also extensively tested in cluster environments [5], however, new I/Os frameworks such as NIO.2 or XNIO3 appeared, and they require more attention at the moment.

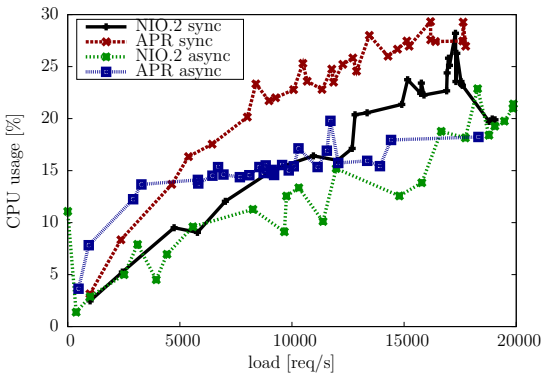


Fig. 9: CPU usage against load in JBoss AS 7, apr vs nio2 tests.

V. Conclusions

In this work, we directly compare NIO.2 and XNIO3 I/Os in terms of different performance metrics. We notice, that while having very similar behavior in the blocking mode, they operate differently upon non-blocking operations. NIO.2 presents low CPU usage and short RTs, whilst XNIO3 optimizes memory consumption. Because of these properties, we select NIO.2 to implement our first connector, but as a future work we plan to develop the XNIO3 implementation to check how low memory consumption influences server performance. We compare the newly developed connector against the existing APR based one. The APR connector is considered as a solution of high performance, but the NIO.2 one outperforms it in terms of both CPU usage and RTs in our experimental setup. Our results are promising, hence we want to use the NIO.2 based connector per default in future releases of the JBoss AS.

References

[1] "Jboss application server 7," 2012. [Online]. Available: https://docs.jboss.org/author/display/AS7/Documentation

[2] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *USENIX Symposium on Internet Technologies and Systems*, 1997, pp. 61–71.

[3] J. Hu, S. Mungee, and D. Schmidt, "Techniques for developing and measuring high performance web servers over high speed networks," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1998, pp. 1222–1231 vol.3.

[4] J. Cane, "Measuring performance of web applications: Empirical techniques and results," in *SoutheastCon, 2004. Proceedings. IEEE*, 2004, pp. 261–270.

[5] R. Ross, D. Nurmi, A. Cheng, and M. Zingale, "A case study in application i/o on linux clusters," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, ser. Supercomputing '01. New York, NY, USA: ACM, 2001, pp. 11–11.